ARMY RESEARCH LABORATORY

ProjectHPC: A Multi-Tier Architecture for Simulation and Analysis

by Jerry Clarke, Kelly Kirk, James Collins, Ankur Chopra, and Kenneth Renard

ARL-RP-0334 **September 2011**

A reprint from the *IEEE Proceedings of the UGC2011*, Portland, Oregon, 20 June 2011.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005

ARL-RP-0334 **September 2011**

ProjectHPC: A Multi-Tier Architecture for Simulation and Analysis

Jerry Clarke, Kelly Kirk, and James Collins
Computational and Information Sciences Directorate, ARL

Ankur Chopra
High Performance Technologies, Inc

Kenneth Renard
WareOnEarth Communications, Inc

A reprint from the *IEEE Proceedings of the UGC2011*, Portland, Oregon, 20 June 2011.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2011		

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
ProjectHPC: A Multi-Tier Architecture for Simulation and Analysis	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
Jerry Clarke, Kelly Kirk, James Collins, Ankur Chopra, and Kenneth Renard	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: RDRL-CIH-C Aberdeen Proving Ground, MD 21005	ARL-RP-0334

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

A reprint from the *IEEE Proceedings of the UGC2011*, Portland, Oregon, 20 June 2011.

14. ABSTRACT

For Mobile Network Modeling, Multi-Scale Modeling, and Blast Protection for Platforms and Personnel, ARL is developing and providing a variety of codes and tools that require a wide range of computer sophistication from the user. This diverse and complex software supports a range of interfaces that extend from simple, intuitive interfaces with a minimal number of options to scalable, parallel simulators that are submitted via a batch queuing system. Subject matter experts with little to no computer science experience will benefit from the delivered software environment as well as expert users with intimate knowledge of the inner working of a parallel simulator. By definition, high performance computing attempts to attain the most performance from computational resources as possible. Usability is not a priority, which is absolutely as it should be; however, while a huge number of analysts and periodic users could potentially benefit from access to HPC, the "cost of entry" is just too high. Our solution is a multi-tier architecture that provides expert user interfaces for code developers, computer scientist interfaces for sophisticated users, and analyst interfaces for subject matter experts in fields other than computer science.

15. SUBJECT TERMS

Project HPC, multi-tier, portal, Django, MNMI, HPC

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU	18	Jerry Clarke
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER (*Include area code*) (410) 278-9279

ProjectHPC: A Multi-Tier Architecture for Simulation and Analysis

Jerry Clarke, Kelly Kirk and James Collins
US Army Research Laboratory
{Jerry.Clarke, Kelly.T.Kirk, Pat.Collins4}@us.army.mil

Ankur Chopra
High Performance Technologies, Inc
Ankur.Chopra.ctr@us.army.mil

Kenneth Renard
WareOnEarth Communications, Inc
Kenneth.Renard@us.army.mil

Abstract

F*or Mobile Network Modeling, Multi-Scale Modeling, and Blast Protection for Platforms and Personnel, ARL is developing and providing a variety of codes and tools that require a wide range of computer sophistication from the user. This diverse and complex software supports a range of interfaces that extend from simple, intuitive interfaces with a minimal number of options to scalable, parallel simulators that are submitted via a batch queuing system. Subject matter experts with little to no computer science experience will benefit from the delivered software environment as well as expert users with intimate knowledge of the inner working of a parallel simulator. By definition, high performance computing attempts to attain the most performance from computational resources as possible. Usability is not a priority, which is absolutely as it should be; however, while a huge number of analysts and periodic users could potentially benefit from access to HPC, the "cost of entry" is just too high. Our solution is a multi-tier architecture that provides* ***expert user*** *interfaces for code developers,* ***computer scientist*** *interfaces for sophisticated users, and* ***analyst*** *interfaces for subject matter experts in fields other than computer science.*

1. Introduction

In 1963, before ZIP codes were introduced in the US, Glen Culler and Burt Fried described ***An On-Line Computing Center for Scientific Problems*** (Culler and Fried 1963)**.** They lamented about the difficulty scientists and mathematicians have taking full advantage of the "impressive achievements in computer hardware and programming techniques". They wrote:

> *The source of the difficulty is basically the poor communications between the "user" ... and the computer ... as well, upon the inherent difficulty of imparting to a programmer the detailed and specialized knowledge one acquires about a particular problem area after working in it for some time.*

Naturally, their problems were complicated by the fact that their RW-400 computer had only 1024 words of memory and stored functions on an 80,000 word magnetic drum. Their interface was two 17 inch CRT oscilloscopes for graphics and an 8 inch CRT for alpha-numeric output.

So how far have we progressed from those heady days when the favorite programming language was solder? Even after the introduction of the Graphical User Interface, Windowing Systems, Client-Server Software Architectures, Web Services, Grid Computing and more, many of today"s scientists and mathematicians would lament the same problems. In fact, the gap between "user" and "programmer" is widening. Effectively programming today"s parallel systems with tens of thousands of cores and providing a simple to use interface to the desktop is challenging for even the most skilled programming teams.

During the mid 1990s and early 2000s many meta-computing tools were available (and some still exist) that allowed the computer scientist to design complex distributed computing systems providing a variety of reusable layers. Most of these were designed by computer scientists, for computer scientists. Indeed, examining system diagrams and API specification for many of these is daunting. Some of these tools relevant to meta-computing applications included (Allan and Ashworth 2001): Legion, GLOBUS, AppLeS, CORBA, Nimrod, NetSolve, Synthetix, Chorus, InfoSpheres, Amoeba, MILAN, Arjuna, Apertos, GrassHopper, WAVE, Locust, HPVM, HPC++, CC++, MIST, GA, Fortran-M, HPF, Java, Raja/RMI, Jini, ANDF, DQS, NQS, LSF, Condor, NQE, LoadLeveler and Cumulvs. These tools helped address some of the technical challenges and architectural issues involved in designing systems that can bring the full power of today"s computing systems to users that are not computer scientists. Technical challenges, however, are only a piece of the overall problem. Local policy restrictions, particularly with computer security are a major concern. Additionally, systems that "*make HPC easier to use*" still require an in depth knowledge of the target application, HPC platforms, or both.

Today, multi-tier architectures have graduated from the halls of computer science departments and have become widely used in the business community. They provide separation of the data, presentation, and business logic layers making it possible to design robust, maintainable application specific systems that are used by subject matter experts in fields like accounting and medicine. They provide the scalability, flexibility and modularity that is difficult to reproduce in simple client-server architectures. They provide an excellent model for deploying systems of interest to the DOD that deliver a top-down solution to a scientific problem as opposed to a bottom-up solution that makes HPC easier to use.

Recent advancements in Python communication and Web frameworks, as well as advancements in JavaScript toolkits have made developing and deploying multi-tier architectures easier. So, building upon the widely used *Computational Science Environment* (Clarke, et al. 2010) ***(CSE)*** the US Army Research Laboratory is developing and deploying ProjectHPC. ProjectHPC is a multi-tier architecture that provides ***expert*** interfaces for code developers, ***computer scientist*** interfaces for sophisticated users, and ***analyst*** interfaces for subject matter experts in fields other than computer science. Maintaining multiple interfaces for various levels of expertise is imperative since exposing all functionality at all levels is counterproductive. This is similar in concept to various Linux distributions, where usability approaches the familiar Microsoft Windows (Holmes and Cox 2011) for the general user community without removing the command line interface beloved by experts.

The goal of ProjectHPC is ***not*** to design a generalized framework and service provider like nanoHUB (Klimeck, et al. 2006), rather it is to acquire and/or develop tools that take advantage of advancements in JavaScript toolkits and Python frameworks to deliver targeted interfaces to complex, distributed functionality. Initially, the focus is in three primary scientific areas: Multi-Scale Modeling, Mobile Network Modeling, and Blast Protection for Platforms and Personnel. While not inhibiting the expert user or computer scientist, ProjectHPC endeavors to deliver an interface to the analyst or subject matter expert that is focused on problem solving, thus relieving the user from the complex tasks of marshaling the proper resources and managing the complex interactions between parallel computer simulations, data management, and a responsive user interface.

2. Computational Science Environment

The Computational Science Environment is a python aware environment developed around the common data model using the eXtensible Data Model and Format (XDMF). It provides a standard environment for data analysis, visualization, and software testing and evaluation. It incorporates many industry standard software development APIs, cross-platform scripting languages, a large number of available analysis and visualization libraries and tools, code control repositories, a build system, and much more. It is the foundation for ProjectHPC.

Xdmf provides the glue for CSE. It ties together scientific application codes, data analysis codes, and visualization tools in multiple ways. Codes that output their data in Xdmf format can take advantage of visualization tools such as ParaView and Ensight. It also provides mechanisms for stand alone, scientific application codes to share data during the course of a calculation, thus producing coupled calculations. The sharing of data can be through memory or disk, and on the fly. Xdmf is flexible. One example is sharing data at well-defined simulation times between CFD and CSM codes.

ProjectHPC builds upon the successes of CSE by taking advantage of Xdmf and the many analysis tools and the rich development environment it provides. The core applications of ProjectHPC define an API for interfacing scientific

application codes. Using this API, application developers can quickly build web-based tools for solving complex engineering problems and can take advantage of the many tools CSE has to offer. ProjectHPC will eventually be bundled with CSE and made widely available.

3. ProjectHPC

3.1 Architecture

ProjectHPC is a multi-tier architecture consisting of three main tiers (or layers): a User Interface Layer (UIL), a Logic Layer (LL), and a Compute Layer (CL), see Figure 1. The Compute Layer is not formally part of ProjectHPC; but, because it is central to the system we refer to it as the third tier of the architecture. As mentioned in the Introduction, there are multiple interfaces into this architecture serving a variety of user needs. These interfaces are shown schematically in Figure 1 and described below.

The User Interface Layer provides the "Analyst" and "Computer Scientist" interfaces into the system. The "Analyst" interface is web based and gives access to workflows that solve specific DoD problems. This is for analysts who are only interested in results and do not have the time or organization mandate to become HPC experts. To use the system an analyst interacts with the web page by selecting a problem, providing input and initiating tasks. Upon completion of the tasks results are presented to the user in some useful format.

A second interface is provided for computer scientists. This interface gives access to the framework and is for users who need more flexibility than the web interface provides. Computer scientists will have access to scripts, frameworks, and libraries that facilitate data manipulation, data modeling and visualization. They can take advantage of this layer by building web or other interfaces into workflows, or building additional application logic as necessary. A workflow defines a solution to a particular problem. For example, a workflow might consist of integrated simulations from multiple disciplines, such as CSM, CFD, FMS, etc, just to solve a single problem. A major goal of ProjectHPC is to construct a framework so that workflows that solve problems of interest to DoD can be easily built and used. Computer scientists working with application code developers will be able to rapidly develop web interfaces to workflows that help design the next generation military systems.

The Logic Layer is the middleware between the user interface layer and the compute layer. There are two main parts to this tier: the core logic and application logic. Core logic implements basic user, task, file, authentication and data management services. It also manages the interface into the Compute Layer. An API into the core logic is well defined. Application logic takes advantage of this API by implementing specific workflows of interest to end-users. At this layer inputs are processed from the UIL, and associated tasks are created and managed. These tasks may include submitting jobs to HPC systems and providing the results back the UIL.

The Compute Layer provides the computational resources necessary for solving many problems of interest to DoD scientists and engineers. Because of this it is a critical layer in the multi-tier architecture. While not formally part of the architecture, ProjectHPC"s core logic will provide the hooks into the large HPCMP HPC systems and an API to build interfaces to other Linux systems.

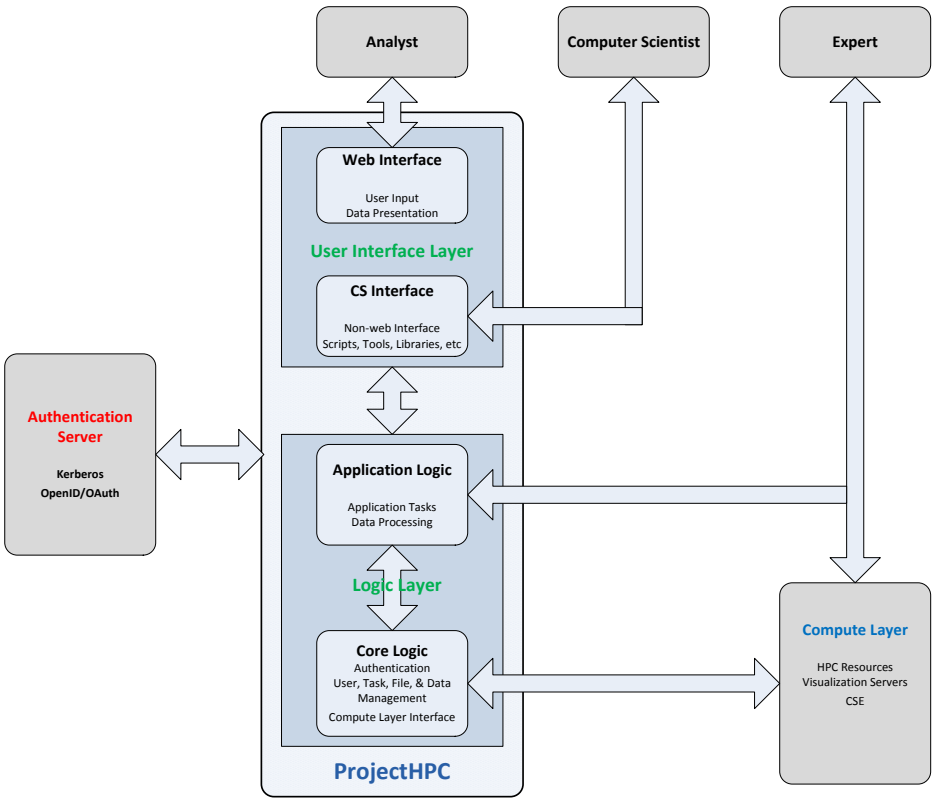

Figure 1 ProjectHPC Architecture

3.2 System Design

Web Server

In simple terms, the front end web-server's job is to listen on port 80 of the machine – or whatever port has been designated for the website – and serve static and dynamic content to the client.

> **Static content** – the web-server delivers exactly what is stored on the server machine
> **Dynamic content** – the web-server prepares the content with fresh information at request-time using a server-side scripting language

Since python was chosen as the server-side language of choice, we need to be able to link the web-server with Python. mod_wsgi is an Apache module that does exactly that. It can host any Python application that supports the Python WSGI (Web Server Gateway Interface) is well suited for hosting high performance production web-site.

> **WSGI** – Python's Web Server Gateway Interface is a specification for web servers and application servers to communicate with web applications.

mod_wsgi has two primary modes of operation. In „embedded" mode, mod_wsgi works such that the python application code will be executed within the context of the normal Apache child processes, much like hosting PHP applications. The alternate and superior method of operation, available only with Apache 2 on UNIX, is the daemon mode. Daemon mode operates similarly to FastCGI but is less complex, whereby distinct processes can be dedicated to run a WSGI application, much reducing the impact of serving up content on Apache child processes. While it is true that

embedded mode can technically perform better than daemon mode, to get the best performance from embedded mode will require fine-tuning the Apache Multi-Processing Modules (MPM) settings, which are biased towards serving static media and PHP applications. Daemon mode is thus the preferred mode of operation for mod_wsgi.

Web Frameworks

In the not so distant past, web development solely involved client side computation and rendering. HTML, CSS and JavaScript were the only resources available. With the advent of server-side scripting languages like PHP and ASP, web infrastructure development has seen tremendous growth. Today, the web involves not only writing web-pages but also databases, authentication, caching, testing, deployment, security, monitoring, API's, search, SEO, task queues, parallel computing, just to name a few. In order to manage the web infrastructure effectively, establish standards, promote reusable codes and most importantly, make web development easy again it is necessary to use Web Frameworks.

> A **web framework** is a collection of packages or modules which allow developers to write Web applications or services without having to handle such low-level details as protocols, sockets or process/thread management (Python Software Foundation 2011).

Frameworks provide abstractions for a number of core activities such as interpreting requests, processing requests, storing data and producing responses. While a majority of web frameworks are still exclusively server-side technology, some web frameworks are now focusing on the client-side as well, thus, attempting to provide a complete solution for application development. For a developer, writing code for a framework typically involves conforming to conventions that let you "plug in" your code into the framework, thus concentrating on the logic of the application instead of worrying about the server-side infrastructure.

Django – A Python Web Framework.

Django is a "high-level Python Web framework that encourages rapid development and clean, pragmatic design" (Django Software Foundation 2011) with claims to be the go-to "web framework for perfectionists (with deadlines)." Leaving the marketing speak aside, django does in-fact have a lot going for it.

Django was developed by a "fast-moving online-news operation" to build high-performing yet elegant Web applications under the intensive deadlines of a newsroom. Moreover, it was subject to heavy use for over two years in a production environment before being open sourced in 2005. The intensions were clear from the start - to create a fast and easy framework that focuses on automating as much as possible and adheres to the Don't Repeat Yourself (DRY) principle.

Django's major advantage over its competition is its performance. Every bit of the framework is designed with efficiency in mind (boo"s from the Ruby-on-Rails crowd?). Django is robust and scales very well. Sites will not go down, even when attacked (slashdottings, farkings, anyone?). Django's object-relational mapping (ORM) is designed to perform the least possible database lookups and its template system is extremely fast, so fast in-fact that Jacob Kaplon-Moss (Jacob Kaplan-Moss 2011) claims that in their testing caching compiled templates was actually slower then re-rendering them on every request; there are also some unverified benchmarks that suggest that django"s template language is about four times faster than Kid (Kid 2011) and about six times as fast as Chettah (Cheetah 2010). In addition to this, Jacob claims that he serves over 15 million hits a day to about ten sites using a single django installation (one web server and one database server). If django is still not fast enough, it is easy to cache performance-intensive bits using cache back-ends like memcached (Memcached 2010).

> **Memcached** – A high-performance, distributed memory object caching system used by websites like Wikipedia, Flickr, Twitter, Youtube, Craigslist, WordPress.com and many more.

While performance is important to django, so is ease of use. Django encourages doing the right thing. In django doing the wrong thing is hard. The most annoying and frustrating part of website development is writing administration interfaces, that is, writing form validation scripts over and over. Well, django gives you a fully-functional automated administration interface (based on the ORM) absolutely free, so that developers can focus on their apps rather than bookkeeping operations. And while there has been some criticism about the django template language not being "pythonic", in our opinion, it rightly separates developers from designers. The template language is similar in usage to

Smarty (a PHP template language) and it should be an easy transition for designers. In addition, django has a mature documentation and release process and a backwards-compatibility policy. But most importantly, django is Python, and gives developers the awesomeness of python (Python powered!), its readable syntax, dynamic typing, high-level OO and cross-platform goodness.

If being fast and easy is not enough, django is secure. It prevents cross-site scripting and SQL injection attacks, and it also checks directory traversal and buffer overflows. Testing is part of the Django culture. Any code not tested is considered broken so django includes a testing environment, and there are many tools available to test codes and produce reports.

Django is a real world framework that solves real world problems. Not only is it good for creating small personal sites on shared hosting system, it also works well in environments ranging from small enterprise sites to large social networking sites with huge databases, all without having to reinvent the wheel. This may well be the reason behind its adoption by organizations like Google, Discovery Communications, The New York Times, PBS, The Washington Post, NASA, National Geographic, MSNBC, Library of Congress and many more.

ProjectHPC – A Django Based Web Application Framework

Reiterating what was said before, for a developer, writing code for a framework typically involves conforming to some kind of conventions that lets you “plug in” your code into the framework. ProjectHPC (P-HPC) subscribes to same idea, but, goes beyond the “plugging in” feature that django has to a “Plug-n-Play” architecture. This means that P-HPC apps (think widgets) require minimal configuration to install and run. P-HPC uses all that is sweet in django and makes it sweeter by building an architecture around the django framework that provides support for designing, modeling, monitoring, executing and optimizating.

P-HPC codes form an application framework (quite different from the django framework), that allows building highly customized, efficient, secure and massively parallel web apps with relative minimal programming effort. As with other web frameworks, your apps will run as long as they conform to certain rules set by framework. It must be noted that P-HPC apps are actually django apps confined under limitations and functionality offered by the P-HPC core. This harness around django apps allows P-HPC to carry out several common operations on framework apps and thus tremendously simplifies automation. For instance, the P-HPC Job Manager can run any job, defined within the framework, without requiring any explicit implementation details, thus providing 'plug-n-play' operation.

We have used the 'plug-n-play' cliche a couple of times already, but what we would like to stress here is that P-HPC apps are independent, such that they are completely self contained and do not interfere with other apps on the same project. They can be added or removed without breaking the system and that all relational dependencies are confined within the app. This modular architecture not only makes it easy to deploy and retire apps, but also simplifies developing, updating, upgrading and testing apps on the framework.

P-HPC is also capable of interacting with users at different levels of computer science expertise. P-HPC operates on two layers - the User Interface (UI) layer and the Logic Layer - and uses a third Compute Layer for task execution. The UI layer is accessible via two distinct interfaces, a web interface for analysts and a Computer Scientist (CS) interface. The CS interface provides computer scientists with the ability to communicate with apps using custom scripts and tools. The Logic layer, on the other hand, constitutes apps of two types, core apps (core logic) and framework apps (application logic). The projects main challenge is to create effective and efficient methods for interaction between adjacent layers, thus allowing the flow of information between the layers. In effect, P-HPC is the glue between the three layers.

P-HPC comprises of a set of *core apps*, these core apps and their API's set the development environment for *framework apps*. At the time of writing, the core comprises of three apps, namely, User Manager, File Manager and Job Manager. The User Manager app is completely integrated with django's inbuilt user authentication system, but it does not modify it in any way, in-fact it extends it to include features like custom user permissions, custom user groups, supplemental user data and support for future OpenID integration. The File Manager app assigns each P-HPC user with a “home directory”, which may be a linking point to the web-server's local file-system or one-or-more remotely located directories. It also provides users the ability to add multiple (hierarchical) files and folders and mimics the UNIX directory-file structure in the P-HPC database to make file access and deletion easy for P-HPC app developers. To be able to run

jobs, the Job Manager first defines what a Job is in P-HPC, sets the spec for extended jobs that are defined in the Framework apps, registers extended jobs for polymorphic access, defines operations each extended job must perform and provides an API for job execution.

A significant amount of care has been taken while designing the P-HPC - django integration portions. The core apps have been designed such that they extend the django code base, thus preventing any alterations to django's base code. This was done to ensure less complicated upgrade paths, maintain security and to keep compatibility with django's fantastic admin interface.

P-HPC, in simple terms, takes job creation and initiation requests from the UI Layer (browser or CS interface) and then processes and tracks the job's several tasks at the Logic Layer using functionality built within the core. Once a job is completed, the results may be accessed at the UI layer. The Compute Layer (HPC or other compute resource) already has well established methods for running tasks (execution units). Each P-HPC job may have one or more smaller tasks, therefore the Logic Layer (core logic) must run tasks at the Compute Layer in the most efficient manner possible and subsequently keep track of the task status so that job results are retrieved as soon as all related tasks have completed execution and the results are available.

P-HPC, as described above, will produce the required results, however it is important to note that each HTTP request in django must return. In short, django cannot defer result retrieval for its requests. Django will keep an HTTP request open until it receives a response or the request times out. While this might not seem like a major problem at first, as traffic grows, maintaining several simultaneous open HTTP requests will severely impact performance. Concurrent operation is a therefore a requirement for P-HPC's success.

Fortunately, there is a solution for allowing concurrent processing of tasks in django. The idea is to maintain a task-queue execution mechanism at the Logic Layer. Tasks are queued on to a message queue that implements the Advanced Message Queuing Protocol (AMQP). A listener then listens to the message queue and delegates processing to available workers. One or more worker servers then concurrently process tasks using multiprocessing. In P-HPC, a project called Celery (Celery 2011) and its django integration piece django-celery handles task processing; Rabbit MQ (Springsource 2011) handles messaging and Kombu (Kumbu 2011) is the listener.

AMQP – An open standard protocol for message orientation, queuing, routing, reliability and security.
RabbitMQ – The leading implementation of AMQP, written in Erlang, a programming language with built in support for concurrency, distribution and fault tolerance. It is a proven platform that offers a reliable, highly available, scalable and portable messaging system with predictable and consistent throughput and latency.
Celery – An asynchronous task queue based on distributed message passing that supports both real-time operation and scheduling. Tasks in celery are executed concurrently on one of more worker server and are able to execute both asynchronously (in the background) or synchronously (wait until ready).
django-celery – Integrates celery with django.
Kombu – AMQP messaging framework for python.

With the concurrent task processing system in place, we now have the ability to queue real time tasks and even schedule tasks, but what if the execution of one task is dependent on the other? When do we queue the dependent task? Even if the dependent task is queued after the main task, the dependent task may reach execution before the main task has executed and this will result in errors. Since Celery does not understand the concept of dependent tasks, the user will have to wait for the main task to complete and only then physically queue the dependent task. This is not practical or efficient. P-HPC therefore provides task management and monitoring utilities in its API's, including a subtask decorator to automate task status checks. A task decorated as a subtask can track (status check) its child tasks and also be tracked by its parent task. This is achieved by storing small bits of information that uniquely identifies each task in RAM using memcached, and passing a list of parent (parent of parent and so on) identifiers to subtasks.

Yes, ProjectHPC is still in the development stage and is rapidly evolving, however all skeptics should be relieved by the fact that everything described in this paper, though not production ready, is functional and has been prototyped. The demo's currently being served by P-HPC are for Mobile Network Modeling and we are also working on Physics Modeling applications with ParaviewWeb integration. P-HPC is capable of and is looking to build further solutions to demonstrate the significance of our Multi-tier Architecture.

ParaView – Paraview is an open-source, multi-platform data analysis and visualization application that allows users to quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities.
ParaViewWeb – ParaView Web aims to provide a collaborative remote web interface for 3D visualization with ParaView as a server.

A P-HPC app implementation that demonstrates the capabilities of the architecture is the NetSim app developed for the Mobile Network Modeling Institute (MNMI).The purpose of this app is to generate impressive and dynamic visualizations for both livedata feeds and NetDMF file playback. Figure 2 and Figure 3 are some screen-shots that demonstrate some of the visualizations running on P-HPC. Figure 2 is a snapshot of a communication matrix taken from a simulation that showed live device communication data from an NS3 (NS-3 Project 2011) simulation.

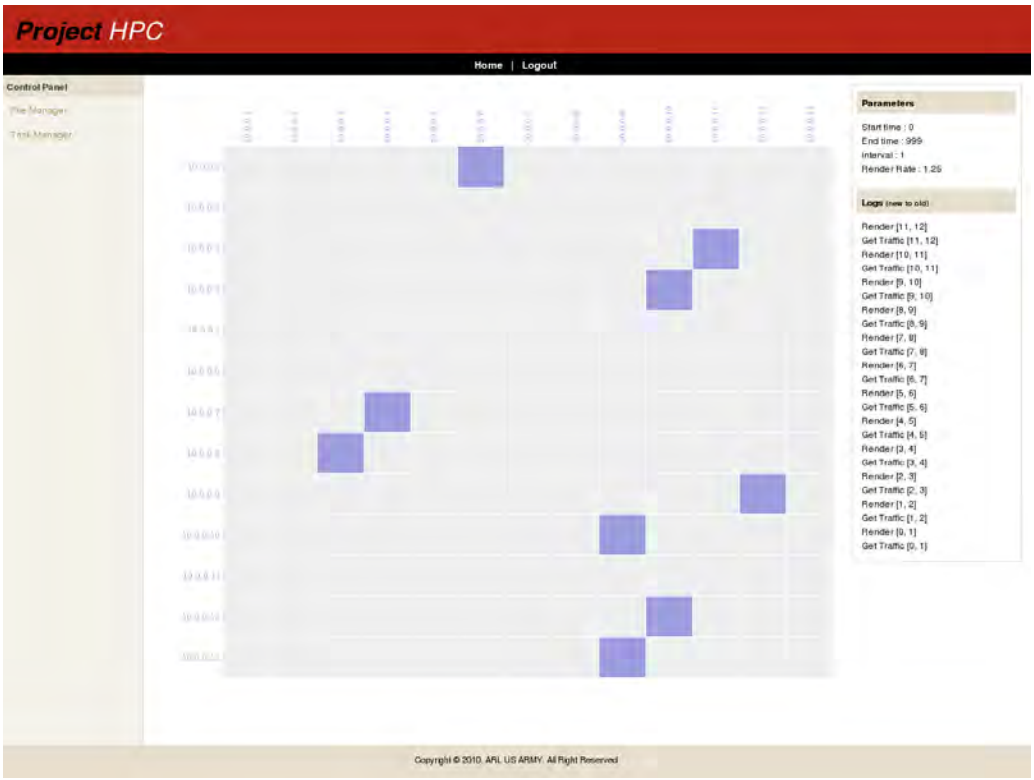

Figure 2 A snapshot of a device communication matrix taken from a live NS3 simulation. The colored boxes indicate communication active links between devices on the x and y axis. Powered by Protovis, JQuery, and ProjectHPC.

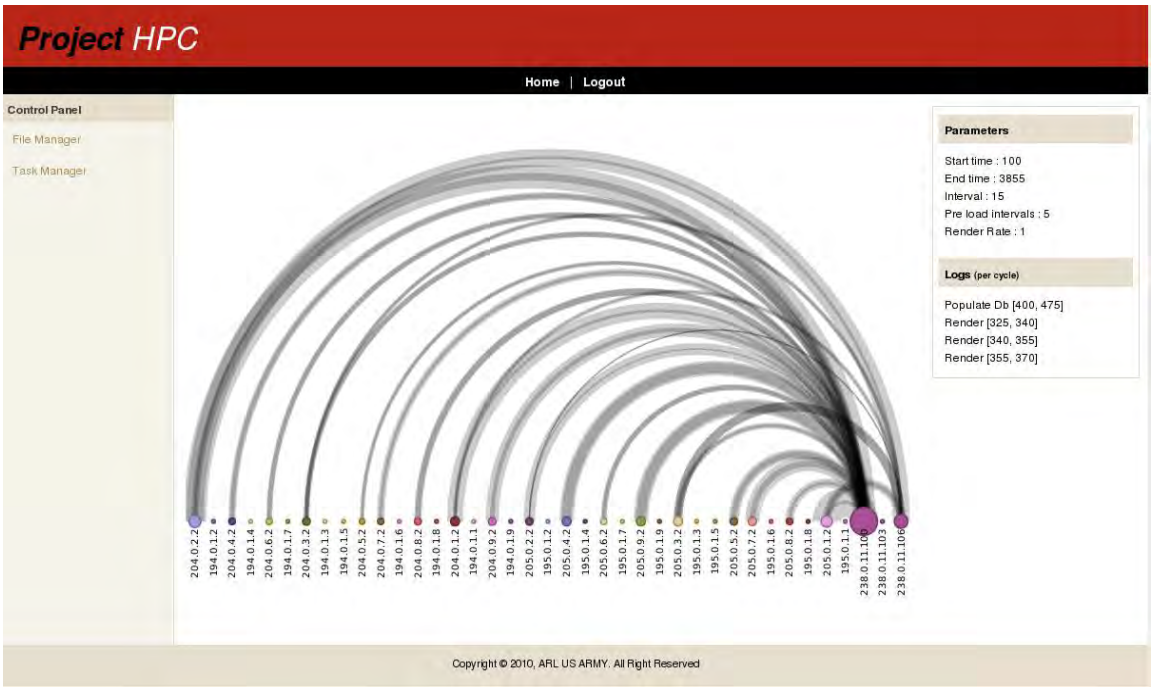

Figure 3 An arc diagram visualization playing an NetDMF communication file. Powered by Protovis, JQuery and ProjectHPC.

Figure 3 shows communication between devices. The width of the connection represents the relative amount of data transferred. The size of the colored balls indicates the amount of data transferred by the device and the darkness of the lines indicates line overlaps.

NetDMF – An extensible Discrete-Event Mobile Networking Data Model and Format
NS3 – A discrete-event network simulator for Internet systems, targeted primarily for research and educational use
Protovis – uses JavaScript and SVG for web-native visualizations
JQuery – a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

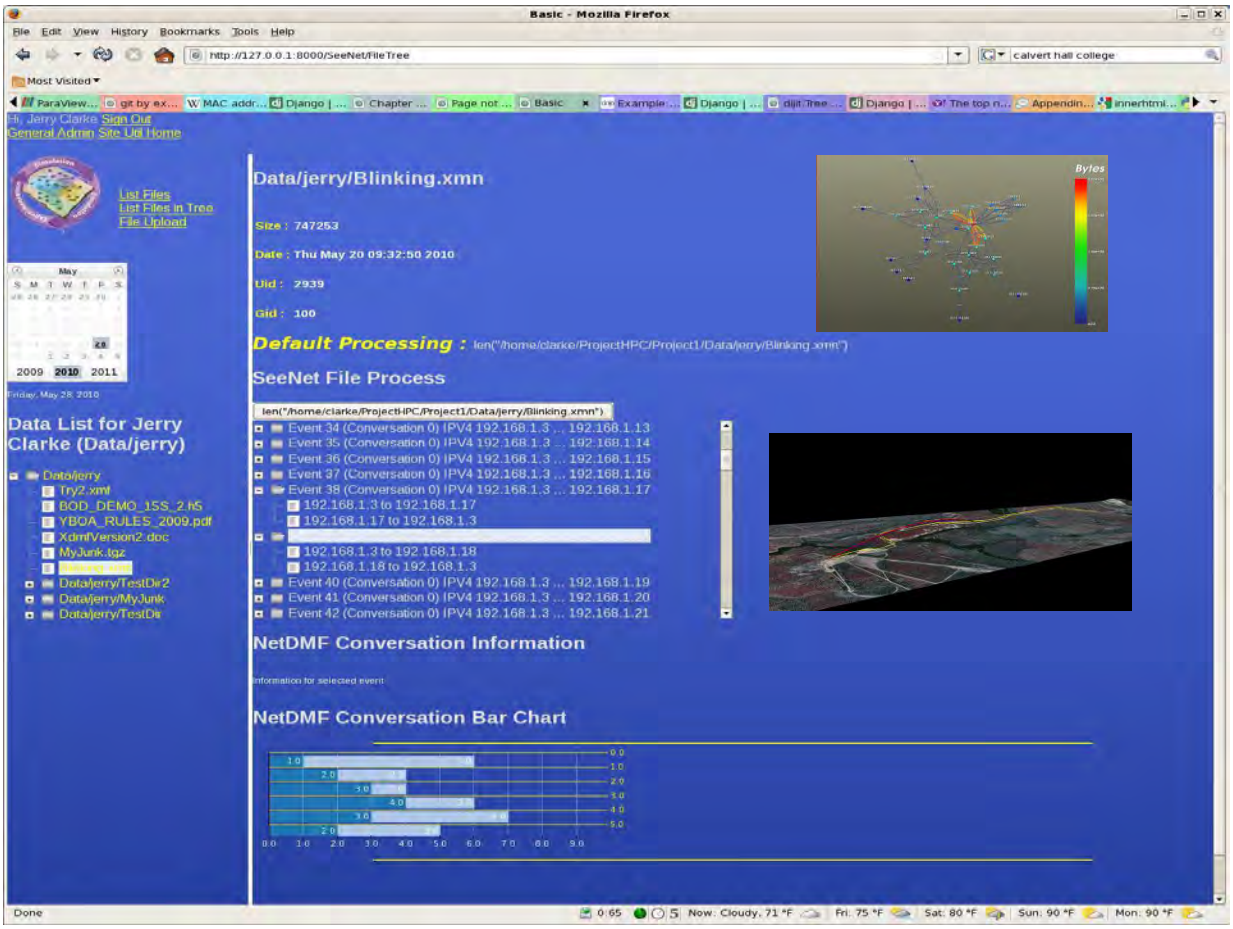

Figure 4 ProjectHPC's vision for the future.

Figures 2 and 3 show some of the capabilities of ProjectHPC as it exists today, but the vision is much greater. A glimpse of the future is presented in Figure 4. It shows a battlefield network simulation launched from a web site where the user has simulation time access to the data and can easily interact with it. This example has a 2D graphic showing connectivity between nodes on a battlefield with color coded data transfer rates, and a 3D graphic showing the terrain. The user will be able to interact with the data as the simulation progresses; in particular, the user will be able to view 3D data from whatever viewpoint that makes sense and make decisions based on the results. Simulations can be controlled with ease, all from a web interface.

ProjectHPC Authentication

Multi-tiered web applications pose additional security challenges since a web server is attempting to execute an action on the back-end tier on behalf of the user. This requires the web server to possess and forward authentication information that originates from the user. Typically, this information includes a secret that is meant to be shared only between the user and the back-end tier. These credentials can be vulnerable to a rouge web server or an impostor claiming to be the legitimate one. Even on the legitimate web server, a collection of credentials from a large set of users becomes a high-value target for attackers. A complicating factor is that most authentication systems do not limit the access granted by the use of their credentials. For example, a password or Kerberos ticket is usually capable of granting full user permissions to do anything that the user is authorized for. During transmission and storage on the web server, protection of these valuable credentials is essential.

Our approach to implementing security measures for ProjectHPC is to start with the current practice as developed and deployed in the User Interface Toolkit (UIT) project. This involves obtaining Kerberos credentials for the user on the web server by passing in the Kerberos principal name, Kerberos password, and a SecurID passcode. Users that do not have

SecurID cards are registered to use HSAM (HPCMP Single-Use Authentication Method), which provides a passcode in place of the SecurID passcode. The Kerberos ticket obtained is first validated against a Kerberos principal known to the web server and is used for local authentication. The Kerberos ticket and secret key are then encrypted and encoded as a cookie that is sent back to the user. The encryption key stays on the server, only to be used by authenticated client connections when needed for authentication to the back-end system. This approach limits the storage of credential information on the server, as well as protects the credential in transit to and from the client.

This project is also looking forward to addressing other challenges of multi-tiered authentication systems by prototyping the use of OAuth as a way to limit credential capabilities, and to avoid sending credential information (passwords, passcodes) to the web server. Currently, OAuth version 1 is being used in prototypes that rely on authentication directly to the back-end tier where the user must specifically authorize access to the back-end tier from the web server. The OAuth protocol flow first establishes a "request token" by authenticating the MTS to the back-end system, and then re-directs the user to the back-end system to authenticate themselves and authorize the pending transaction. Upon successful authentication and authorization, an "access token" is granted to the web server which it uses to authenticate on behalf of the user to the back-end system. The access token is limited in time and in scope based on the decision the user made at the time of authorization. The authentication and authorization steps are meant to protect and limit the authority granted to the web server and be quick and easy for the user to execute.

To that end, authentication to the back-end system could be provided by a PKI authentication or another web-based authentication such as OpenID. We intend to integrate OpenID as an authentication system to this environment as implemented by the HPCMP security infrastructure. Coordination and cooperation with the HPCMP security staff is the foundation of these efforts and can serve as a model for future authentication of web services.

4. Conclusion

ProjectHPC implements an architecture that can provide targeted interfaces to complex software to analysts and subject matter experts who are only interested in results and who do not have the time or organization mandate to become HPC experts. These targeted interfaces fit into the NIST definition for cloud computing (Mell and Grance 2011). It provides convenient, on-demand access to shared applications that solve complex military problems using the “Sofware as a Service” service model. ProjectHPC, however, goes beyond that. ProjectHPC provides multiple interfaces into the architecture. Computer scientists that need more than just web access will have an interface into the system that allows them to take advantage of the framework API to build their own applications and interfaces. Expert users will have access to the application interfaces. In short, ProjectHPC targets the analyst but doesn‟t lose sight of the more sophisticated user‟s needs.

ProjectHPC will eventually be bundled with the Computational Sciences Environment and will be made widely available to the DoD community.

References

Allan, R. J., and M. Ashworth. *A Survey of Distributed Computing, Computational Grid, Meta-computing and Network Information Tools.* Survey, Computational Science and Engineering Department, Daresbury: UKHEC, 2001.

Celery. *Celery - The Distributed Task Queue.* 2011. http://www.celeryproject.com/ (accessed May 13, 2011).

Cheetah. *Cheetah - The Python-Powered Template Engine.* 2010. http://cheetahtemplate.org/ (accessed May 13, 2011).

Clarke, Jerry, et al. "A Common Computational Science Environment for High Performance Computing Centers." *HPCMP User Group Conference.* Chicago, 2010.

Culler, Glen J, and Burton D Fried. "An On-line Computing Center for Scientific Problems." *Proc. 1963 Pacific Computer Conf.* Piscataway, N.J.: IEEE, 1963. 221-242.

Django Software Foundation. *Django, The Web Framework for Perfectionist with Deadlines.* 2011. http://www.djangoproject.com/ (accessed May 13, 2011).

Holmes, Marc, and Simon Cox. "Delivering End-to-End High-Productivity Computing." *Microsoft Architect Journal*, 2011.

Jacob Kaplan-Moss. *Jacob Kaplan-Moss.* 2011. http://jacobian.org/ (accessed May 13, 2011).

Kid. *Kid.* 2011. http://kid-templating.org/ (accessed May 13, 2011).

Klimeck, Gerhard, et al. "NEMO 3-D and nanoHUB:Bridging Research and Education." *IEEE-NANO 2006. Sixth IEEE Conference on.* 2006. 441-444.

Kumbu. *Python Package Index : kumbu 1.1.3.* 2011. http://pypi.python.org/pypi/kombu (accessed May 13, 2011).

Mell, Peter, and Timothy Grance. *The NIST Definition of Cloud Computing (Draft).* Gaithersburg: NIST, 2011.

Memcached. *memcached - A Distributed Memory Object Caching System.* 2010. memcached.org (accessed May 13, 2011).

NS-3 Project. *The ns-3 Network Simulator.* 2011. http://www.nsnam.org/ (accessed May 13, 2011).

Python Software Foundation. *Python Programming Language.* 2011. http://python.org (accessed May 13, 2011).

Springsource. *RabbitMQ - Messaging That Just Works.* 2011. www.rabbitmq.com (accessed May 13, 2011).

1 ADMNSTR
DEFNS TECHL INFO CTR
ATTN DTIC OCP
8725 JOHN J KINGMAN RD STE 0944
FT BELVOIR VA 22060-6218

3 US ARMY RSRCH LAB
ATTN IMNE ALC HRR
MAIL & RECORDS MGMT
ATTN RDRL CIO LL TECHL LIB
ATTN RDRL CIO MT TECHL PUB
ADELPHI MD 20783-1197

39 US ARMY RSRCH LAB
ATTN RDRL CIH
B SHEROKE
R NAMBURU
D THOMPSON
T KENDALL
ATTN RDRL CIH C
B HENZ
D SHIRES
J CLARKE
K KIRK
P CHUNG
S PARK
A CHOPRA
N KOSKI
K RENARD
J CRONE
J VINES
J COLLINS (20 copies)
ATTN RDRL CIH M
M KNOWLES
M MOTSKO
N LAZORISAK
ATTN RDRL CIH S
L BRAINARD
ABERDEEN PROVING GROUND MD
21005

INTENTIONALLY LEFT BLANK